Funk2: A Distributed Processing Language for Reflective Tracing of a Large Critic-Selector Cognitive Architecture

Bo Morgan
MIT Media Lab
20 Ames Street
Cambridge, MA 02139 USA
bo@mit.edu

***Abstract*—We see the field of metareasoning to be the answer to many large organizational problems encountered when putting together an understandable cognitive architecture, capable of commonsense reasoning. In this paper we review the EM1 implementation of the Emotion Machine critic-selector architecture, as well as explain the current progress we have made in redesigning this first version implementation. For this purpose of redesign and large-scale implementation, we have written a novel programming language, Funk2, that focuses on efficient metareasoning and procedural reflection, the keystones of the critic-selector architecture. We present an argument for why the Funk2 programming language lends itself to easing the burden on programmers that prefer to not be restricted to strictly declarative programming paradigms by allowing the learning of critic and selector activation strengths by credit assignment through arbitrary procedural code.**

***Keywords*-common sense reasoning; cognitive architecture; critic-selector architecture; metareasoning; credit assignment; programming language; reflective causal tracing; EM1; Funk2**

I. Closed-loop Control and Learning

There are many artificial intelligence algorithms that provide explanations for how to accomplish goals or gather rewards in a domain. A basic artificial intelligence system consists of three processes: (1) perceptual data are generalized and categorized to learn induced abstract models, (2) abstract models are used to infer expected hypothetical states, i.e. states of future, past, or otherwise "hidden" variables, (3) actions are chosen based on considerations of different action-dependent inferences.

While there are many types of machine learning algorithms that focus on this abstract 3-step closed-loop process of learning to control, the field of meta-cognition [1] focuses on making at least two layers of closed-loop systems. The first closed-loop learning algorithm learns how to deal with the external world, while the second closed-loop learning algorithm perceives the state of the algorithm below. We see meta-cognition as a layering of learning algorithms, such that the second layer algorithm learns from perceiving the activity of the first layer and controls or modifies this first layer. While it may be clear how to trace changes in the perceptual inputs of layer one of the algorithm, it is less than clear how the second layer learner should monitor the changes in the state of the first layer learner.

II. A Review of The Emotion Machine v1.0

One system that implements commonsense reasoning, based on Minsky's Emotion Machine theory of mind [2], is a metareasoning system for correcting faulty plans, called EM1 (Emotion Machine, v1) [3]. EM1 is written in Lisp, using a Prolog extension as the logical resolution tool. EM1 is a layered architecture consisting of reactive, deliberative, and reflective layers. Mental critics are represented as commonsense narratives that result in queries to a collection of different Prolog knowledge bases. The commonsense narratives are given to the system in a Lisp format that is compiled into the knowledge bases as collections of horn clauses. These knowledge bases consist of collections of domain-specific horn clauses that are divided into physical, social, and mental domains of reasoning. On top of this Prolog logical substrate, the Lisp program is organized into layers as a critic-selector model of mind [4]. The narrative plans that are generated by the deliberative layer are executed by a lower-layer, called the reactive layer. Part of the reactive layer of the algorithm is written in C and runs PID control loops in a simulated social two-wheeled inverted pendulum type robot. EM1 demonstrates how a system can use commonsense narratives in order to reason by analogy in order to generate plans. Also, EM1 demonstrates a learning process that is driven by reflective critical recognition of failure. Because of the complexity of the rigid-body physics in the world, sometimes even the most carefully constructed plans fail. EM1 has a layer of reflective critics that debug deliberative narratives as they are being interpreted by using a collection of commonsense narrative debugging critics. Using narratives about social situations, EM1 infers the goals of the other agents in the world given partial knowledge of their visible physical actions. When mistakes are made in this inference process, the failure is recognized reflectively, after the fact. Specific types of debugging responses are implemented for different forms of critical failures. EM1 is a step toward a large and complex commonsense reasoning agent with multiple layers of metareasoning that inspect, control, and debug mental representations.

A. Common Sense Requires Many Ways to Think

The original theory for how to build the EM1 model was published earlier [5] as a more complicated model involving additional layers of self-reflective, self-conscious, and self-ideal critics for further reflective control. The architecture focuses on combining "ways to think", which are connections and activations of constellations of reflective, deliberative, and reactive sets of critics and selectors. Also, the reader is referred to this work for a more detailed description of the critic-selector algorithm. We need an architecture that allows many different ways to think because while a single reasoning method may work for a specific problem, no any one reasoning method works for solving all different types of problems [6]. Perhaps one day we will discover a simple key or mathematical formula for deriving all other intelligence, but let's first focus on building a machine that demonstrates any kind of robust commonsense intelligence before making any simplifying optimizations. What we see as necessary for developing a model of commonsense intelligence is the ability to quickly switch between different representations of problems that allow different reasoning methods to continue where any single method would have gotten stuck. See [7] for an overview of the Panalogy architecture for a more detailed explanation of why multiple representations that invoke different procedures of thought are necessary for building a model of a robust problem solving mind.

B. A Brief Overview of the EM1 Critic-Selector Architecture

Cognitive activity in the EM1 architecture is driven by *mental critics*. These critics respond to patterns in the representations of the physical world as well as to representations of traces of the mental processes of the architecture itself. Figure 1 shows an example of a mental critic in EM1's Critic-L language. These critics are organized into three layers: (1) the reactive layer, (2) the deliberative layer, and (3) the reflective layer. We will now briefly describe the implementations of these three layers.

C. The Reactive Layer

The reactive layer consists of critics that fire in response to patterns in the sensory state. For example, here are a few Critic→Selector pairs used in EM1:

- Difference Between Conditions And Desires → Propose Action By Analogy
- Special Observation → Act Reflexively
- Being Watched By Helpful Agent → Explicitly Demonstrate Intent
- Preconditions Met For Intended Action → Take Action

D. The Cyclical Process of the Deliberative Layer

Deliberation critics, unlike reactive critics, fire in response to hypotheses, which are plans in the form of narratives about agents and actions that could be taken in the world. The deliberative process occurs in the following three-step cycle:

1) Hypotheses are assessed by evaluating deliberative critics.
2) The hypotheses with least potential are forgotten.
3) Critics fire associated selectors which create improved variations of the remaining hypotheses.

Deliberative critics are roughly divided into six types of hypothesis problems: plausibility, importance, cohesiveness, relevance, informativeness, and consistency. A few example deliberative Critic→Selector pairs are:

- Unknown Action Consequence → Hypothesize By Analogy
- Unknown Relation Consequence → Hypothesize By Analogy
- Unknown Motivation → Hypothesize By Analogy
- Sequential Observation Inconsistent With Dependency → Negatively Assess Hypothesis
- Observes Opposite Of Actor Desire → Make A Note For Reflective Layer
- Actor Causes Problem For Itself → Negatively Assess Hypothesis
- Other Actor Undoes Desire → Make A Note For Reflective Layer
- Implication Not Inferred → Add Implication
- Involves Undesirable Situation → Prepend Repair
- Involves Undesirable Situation → Prepend Repair

E. The Reflective Layer

Reasoning by applying deliberative critics is neither sound nor complete, which leads to errors such as incorrect inferences being made and correct inferences failing to be made. Thus the process of criticizing hypotheses by deliberative critics is itself criticized by reflective critics. The deliberation cycle in EM1 leaves a detailed trace of activity, but this activity is usually difficult to look at, so there are special auxiliary predicates that make it easier to look at. For example, some of these convenience predicates are as follows:

- (asserted-by FACT CRITICISM)
- (ultimately-asserted-by FACT CRITICISM)
- (asserts FACT)
- (engages CRITIC)
- (called-by C1 C2)
- (hypothesis-created-by HYP CRITICISM)
- (opinion-changed-about R S O)
- (narrative-not-used ACTION NARR)

Two example reflective critic→selector pairs in EM1 are as follows:

- action-failed-to-achieve-effect-and-neglected-required-precondition→append-critic.
- partner-not-helping-and-other-failed-to-infer-goal→credit-assignment.

```
(defcritic (reactive*difference-between-conditions-and-desires=>propose-action N)
  (in conditions current-conditions
    (observes :actor ACTOR :prop (not (REL :subject SUBJ :object OBJ)))
    (desires :actor ACTOR
             :prop (observes :actor ACTOR
                             :prop (REL :subject SUBJ :object OBJ))))
  (in narratives N
    (sequential
      (observes :actor ACTOR2 :prop (not (REL :subject SUBJ2 :object OBJ2)))
      (does :actor ACTOR2
            :prop (ACTION :actor ACTOR2 :object SUBJ2 :target OBJ2) [1])
      (observes :actor ACTOR2 :prop (REL :subject SUBJ2 :object OBJ2) [2]))
    (causes [1] [2]))
(=>)
(in conditions current-conditions
  (assert
    (intends :actor ACTOR
             :prop (ACTION :actor ACTOR :object SUBJ :target OBJ) [[S]]))
  (assert (subsit current-conditions [[S]])))
```

Figure 1. An example of an EM1 reactive mental critic in EM1's Critic-L language. The top of the critic is the knowledge base (KB) name and declarative pattern to be matched, while the bottom of the critic contains the KB name and declarative pattern to assert.

III. Funk2 is a Language Specifically Designed for the Next Emotion Machine

Funk2 is designed with the following programming language and cognitive architectural goals in mind:

1) causal tracing of arbitrary lisp-like processes,
2) massively concurrent multi-threaded simulations, including many thousands of lightweight parallel processes,
3) layers of reflective critic-selector problem solving,
4) commonsense reasoning cognitive architecture primitives, such as person, event, goal, belief, narrative, etc.

We have completed goals one and two in our current described implementation. We are currently working on goals three and four.

Just after Push Singh completed his PhD work of building EM1 in 2006, he sadly passed away, leaving a large project filled with great potential for future research, but since 2006, no one to our knowledge has worked on any other critic-selector cognitive architectures related to this work. Because of this, there seems to us to be a hole in this field of reflective critic-selector cognitive architecture research. Two years ago this fact was very clear to us, and we at that point volunteered for the job of rewriting the original architecture in order to pursue some of the original goals. We have programmed the beginning of a new Emotion Machine critic-selector architecture. As there are many future directions for the EM1 architecture, there were also many problems with the architecture as it was implemented then. Some of the primary concerns that we have with this original implementation are as follows:

- Reasoning is very slow with only 21 narratives in the memory of the architecture.
- Critics and selectors are specified in a declarative logical form, which does not allow for reasoning over noisy or imperfect data.
- The declarative form of the Critic-L language allows for inserted procedural Lisp code, but any procedural code inserted in this way is not reflectively traced.
- Critics and selectors cannot run in parallel, which eliminates the potential for using the architecture for solving any form of parallel control problem.
- The architecture has great potential for parallelization but cannot take advantage of multi-core CPUs or distributed processors.
- Architecture is based on the expensive proprietary commercial languages of Allegro Lisp and Allegro Prolog, which cannot be used freely by researchers.
- Self-reflective, self-conscious, and self-ideal critics and selectors are future work to still be implemented.
- Critics and selectors are not learned from experience.

A. Tracing Procedural Branches in Causal Context

Figure 2 shows a schematic example of how a plan interpreter scans along a plan and, whenever it encounters a conditional branch, leaves a causal trace of reasons for each decision made by the process interpreting the plan.

Causality refers to an agent that causes a change in the world. In our case, the agent is vague and consists of the current context of the computation; for example, what goals are currently actively being pursued. This type of memory is important to keep track of in many different representations

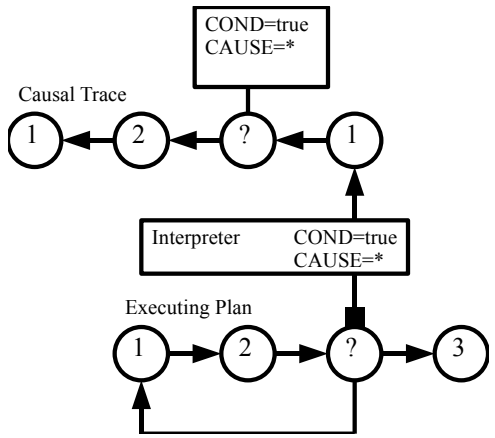

Figure 2. An example of an executing plan containing a loop being interpreted and leaving a trace of plan events with conditional events tagged with contextual information, including the condition evaluated (COND) as well as the causal context (CAUSE). The symbol "*" refers to a list of goals, activated critics and selectors, the function call stack, and other user-specified contextual hints. The "?" represents a conditional expression in a "while" loop.

itself, so that when a change is recognized, perhaps long after the fact, as a bug, then this information can be used for considering multiple different debugging strategies, each associated with a different representation of the cause. For example, all decisions in any computer program can be thought of as a machine code instruction that checks whether specific register is equal to zero in order to decide whether or not to jump to another part of the program. All decisions in a computer are executed at this level of detail, but this is not a good representation for high-level thinking about causality. For most of our causal representations we would prefer a more abstract description of the situation or event in which the decision took place, what agent or agency within the mind was responsible for the code that made the decision. In any case, whether we are debugging a low-level or high-level idea, we prefer to err on the side of being able to record more information than less. Causal tracing can always be set to only trace specific layers of abstraction in Funk2, so machine code tracing is only relevant if the system wanted to reflectively learn something about it's own low-level implementation. Typically, we imagine users of Funk2 mainly tracing and building critics to recognize patterns in their own user-specified contextual hints. There are many other parts of a causal relationship that we would like to be able to refer to in a reflective process that is debugging a bug in a plan:

- Event or situation.
- Agents or agencies involved.
- Knowledge used for compiling those agents.
- Critics and selectors that were active during the compiling of this decision point.
- Reflective processes that were monitoring the planning process at the time.
- Self-reflective critics and selectors that were being used to control the reflective focus.
- Configurations of self-models activated or suppressed during plan creation.
- Goals that were active.
- Self-conscious critics and imprimers[1] that were active, if any.

This a short list of the types of causal knowledge representations that must be handled by different types of causal tracing critic agents.

IV. Funk2 Represents each Critic and Selector as a Separate Thread

Funk2 is an abstract simulation of a distributed operating system. In addition to the normal primitive data-types of most popular languages, Funk2 also allows access to the following primitive data-types as first class citizens:

- cause
- funktion
- object-type
- fiber (virtual thread)
- scheduler
- processor

Also, like many modern languages, Funk2 represents all primitive and abstract data within the language as frames with named and typed slots. Built-in slot types for frames include *get*, *set*, and *execute*. The object system is relatively primitive thus far, without incorporating much of the meta-object protocol [8], but we expect our unique *cause* object to take care of most of the previously complicated meta-object protocol by putting the protocol into the evaluator. The Funk2 core is written entirely in C, just like all popular operating systems these days, so any operating system or external package specific extensions to the language should be easy additions.

A. The Cause Object

Cause objects are created and linked to parent cause objects with the creation of every new execution event, such as the spawning of a new thread. Typically, tracing of multi-threaded programs gets complicated because of the complex inheritance structure of causes for events. Because we have designated a special register in each thread for a cause object, which monitors and controls memory access, we expect to handle this problem more efficiently by always having this abstract control at the locus of every memory access. Without causal tracing of memory access invoked, our interpreter runs at full speed, while when a piece of memory that requires causal tracing encounters a traced

[1] **imprimer:** a mental simulation of a role-model.

cause, this will create events that are appended to a cause-specific trace. We expect cause objects to be one simplifying key to many complicating problems of credit assignment.

Cause objects are frames with typed slots, just like all data in Funk2, but because cause objects follow the causal execution paths of processes, these objects can be used for storing different types of traces of process executions. Funk2 is a reflective frame-based programming language, in which case, "an object would not only represent information about the thing in the domain it represents, but also about (the implementation and interpretation of) the object itself: when is it created? by whom is it created? what constraints does it have to fulfill? etc." [9].

V. Implementation of Mental Resources as Funk2 Fibers

We refer to critics and selectors as types of "mental resources" in our cognitive architecture. As previously discussed, in the EM1 Critic-L implementation of the Emotion Machine architecture, these resources are specified in a lisp declarative form. Because we wish our architecture to remain representationally agnostic and thus more generally applicable for a variety of programmers working together, our mental resources are implemented using any form of the lisp-like Funk2 code. For example, for every mental resource, we allocate a virtual process that we call a fiber; a fiber is to a Funk2 program as a thread is to a C program. Fibers are scheduled by the Funk2 core and are very lightweight bytecode interpreters that execute in parallel. All processes in Funk2 are executed within fibers. Funk2 allows intricate control and inspection of fibers that allow pausing, resuming, and rescheduling fibers. Each mental resource, such as a critic or a selector, is allocated a separate fiber that is initially paused. The architecture supports thousands of concurrent mental resources that can be activated and suppressed by one another. We imagine that most mental resources in a given critic-selector architecture are not active at any given moment. Each mental resource can execute arbitrary lisp-like Funk2 code, given that it is in an activated state. Further, all data that is manipulated by a fiber is manipulated within the context of the current local cause that has activated the mental resource.

VI. How Funk2 Causal Tracing Helps Learning with Critics and Selectors

Figure 3 shows roughly how information flows within and between two of the six layers of the Emotion Machine critic-selector architecture. Unlike in EM1, we expect the next version of this architecture to allow arbitrary procedural descriptions of critic and selector agents, rather than strict declarative logical forms. The added causal memory tracing features of the Funk2 programming language allow credit assignment algorithms to still trace causal agency between critics and selectors. Funk2 keeps track of many types of

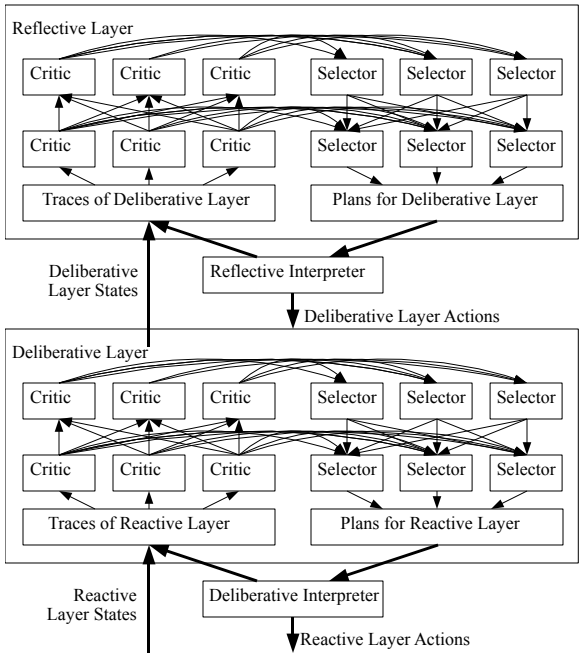

Figure 3. This diagram shows roughly how information flows within and between two of the six layers discussed in the Emotion Machine critic-selector architecture.

primitive memory events for all objects in the language, which allows programmers the freedom to express themselves in a way that is natural for someone familiar with languages such as Python, Java, C++, or Lisp. Funk2 inherits the object-oriented ideas from Java, C++, and Python, such as dynamic object frames and object types, while also incorporating the interactive compiler and "data as code" minimalist syntax ideas from Lisp. While we expect to incorporate a few logical programming ideas from Prolog, we will focus our approach to problem of deduction, induction, and inference from more of a commonsense natural language perspective [10].

VII. Conclusion

So, in conclusion, we see the field of metareasoning to be the answer to many of the large organizational problems of putting together the field of artificial intelligence into one understandable architecture. In this paper we have reviewed the EM1 implementation of the Emotion Machine critic-selector architecture, as well as explained the current progress in redesigning this first version implementation into the novel Funk2 programming language that has been designed for this specific purpose. We have presented a brief review of the recent advances in the machine learning field of credit assignment. We have presented an argument for why we think the Funk2 programming language lends itself to easing the burden on programmers that prefer to not be restricted to strictly declarative programming paradigms

by allowing learning by credit assignment through arbitrary procedural code.

ACKNOWLEDGMENT

This work is supported in part by a Media Lab research fellowship from Bank of America.

REFERENCES

[1] M. Cox and A. Raja, "Metareasoning: A manifesto," *BBN Technical*, 2007.

[2] M. Minsky, *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. New York, New York: Simon & Schuster, 2006.

[3] P. Singh, "EM-ONE: an architecture for reflective commonsense thinking," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[4] M. Minsky, *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind—Thinking, The Critic Selector Model of Mind.* New York, New York: Simon & Schuster, 2006, chapter 7.2, pp. 220–224.

[5] P. Singh and M. Minsky, "An Architecture for Combining Ways to Think," 2003.

[6] J. McCarthy, M. Minsky, A. Sloman, L. Gong, T. A. Lau, L. Morgenstern, E. T. Mueller, D. Riecken, M. Singh, and P. Singh, "An architecture of diversity for commonsense reasoning," *IBM Systems Journal*, vol. 41, no. 3, 2002.

[7] P. Singh and M. Minsky, "An architecture for cognitive diversity," *Visions of mind: architectures for cognition and affect*, p. 312, 2005.

[8] G. Kiczales, D. Bobrow, and J. des Rivieres, *The art of the metaobject protocol*. MIT press, 1999.

[9] P. Maes, "Issues in computational reflection," in *Meta-level architectures and reflection*. North-Holland, 1988, pp. 21–35.

[10] H. Liu and P. Singh, "Commonsense reasoning in and over natural language," *Lecture Notes in Computer Science*, pp. 293–306, 2004.